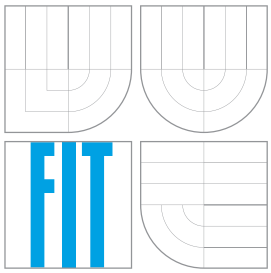
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

JAZYKY PETRIHO SÍTÍ A JEJICH AKCEPTORY

PETRI NET LANGUAGES AND THEIR ACCEPTORS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

IGOR PAVLŮ

VEDOUCÍ PRÁCE
SUPERVISOR

prof. RNDr. MILAN ČEŠKA, CSc.

BRNO 2013

Abstrakt

Tato práce se zabývá P/T Petriho sítěmi a jazyky Petriho sítí. V práci jsou také uvažovány ω–jazyky a jejich vztak k Petriho sítím. Hlavní náplní práce je navržení akceptoru (rozšířené Petriho sítě) některých tříd jazyků Petriho sítí včetně implementace akceptoru těchto tříd jazyků. V práci jsou popsány algoritmy použité při implementaci (rozlišení typu jazyka, automarické vyhodnocení přijetí řetězce, generování řetězců). Na testech je demonstrována správná funkce akceptoru.

Abstract

In this document P/T Petri nets and Petri nets languages are described. In this document ω–languages and relationship of ω–languages and Petri nets is also described. Main focus is set on design of acceptor (extended Petri net) of some classes of Petri nets languages and implementation of acceptor of this classes. Algorithms (recognition of language type, automatical accepting of string, strings generating) used in implementation of acceptor are described. Correct function of acceptor is demonstrated on tests.

Klíčová slova

Petriho sítě, jazyk Petriho sítě, omega–jazyk, akceptor jazyka

Keywords

Petri nets, Petri net language, omega–language, language acceptor

Citace

Igor Pavlů: Jazyky Petriho sítí a jejich akceptory, bakalářská práce, Brno, FIT VUT v Brně, 2013

Jazyky Petriho sítí a jejich akceptory

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením prof. RNDr. Milana Češky, CSc.

.....
Igor Pavlů
14. května 2013

Poděkování

Děkuji podporující rodině a především prof. Milanu Češkovi za jeho vedení, rady a vynaložený čas a trpělivost při konzultacích k vypracování této práce.

Obsah

Kapitola 1

Úvod

Petriho sítě dnes představují jeden z nejrozšířenějších diskrétních matematických modelů pro popis zejména paralelních a distribuovaných systémů. Dokáží zachytit vazby, řídící i datové toky a závislosti mezi prvky popisovaných systémů. Poprvé byly představeny roku 1962 v disertační práci "*Kommunikation mit Automaten*"německého matematika C. A. Petriho a od té doby se jim dostalo mnoha rozšíření a modifikací a jejich studium pokračuje i v současné době [3].

Petriho sítě mají v dnešní době široké uplatnění. Vedle analýzy, návrhu a modelování paralelních a distribuovaných systémů jako jsou databázové systémy, popis komunikačních protokolů nebo paralelních architektur, najdeme uplatnění i v mnoha odvětvých průmyslu nebo při simulaci workflow různých odětvý např. ve státní správě.

Při popisu systému Petriho sítí, získáváme prostředek, který nám umožňuje provádět popis asynchroních událostí paralelních činností. Díky schopnosti abstrahovat určité informace, Petriho sítě umožňují aplikovat metodiku návrhu postupného snížení (zvýšení) úrovně abstrakce.

Vedle modelovacích schopností se studium Petriho sítí zaměřuje na jejich analýzu a vyjadřovací schopnosti, zejména pak analýzu stavového prostoru, strukturálních vlastností a výpočetních posloupností, které jsou generovány těmito sítěmi. Získané poznatky se aplikují při validaci a verifikaci paralelních a distribuovaných systémů.

Velkým přínosem pro rozvoj Petriho sítí byl rozvoj grafických editorů a simulátorů, které umožňují snadnější vytváření modelů formou Petriho sítí a tím usnadňují a zrychlují jejich analýzu.

Formální jazyky jsou jedím ze základních prostředků pro formální popis modelů systémů a ve spojení s Petriho sítěmi nebo automaty představují mocný nástroj pro vytváření modelů chování těchto systémů. S využitím formálních jazyků můžeme rovněž systémy analizovat a verifikovat jejich chování, což může na základě takto získaných poznatků vést až k syntéze systému. I z tohoto důvodu je vhodné zkoumat jazyky Petriho sítí a jejich vztah vůči ostatním typům jazyků v rámci Chomského hierarchie.

V tomto textu se zaměříme zejména na P/T Petriho sítě (Place/Transition Petri nets), které jsou dostatečně obecné a názorné pro výklad.

V kapitole 2 uvedeme přehled pojmů z oblasti formálních jazyků [6, 8] a operací nad jazyky, které budeme v dalším textu využívat. Samostatná podkapitola je pak věnována obecným informací týkajících se třídy ω–jazyků [11, 9].

Popis Petriho sítí a jejich jazyků [5, 4, 7] uvedeme v kapitole 3. Zvláštní podkapitola je věnována vztahu Petriho sítí k ω–jazykům [14, 12], kde si ukážeme, způsob jakým mohou Petriho sítě specifikovat ω–jazyky. Popis návrhu akceptoru, implementace a ovládání akceptoru je obsažen v kapitole 4, kde se rovněž blíže seznámíme s některými algoritmy, které byly použity pro správný běh výsledné aplikace. Ověření funkčnosti společně s ukázkami testů a některými měřeními týkajícími se funkce akceptoru obsahuje kapitola 5. V závěru si shrneme cíle a výsledky práce a další možné směry výzkumu a vývoje. V přílohách nalezneme výstupy některých testů, přehled použitých symbolů a zkratek a obsah přiloženého CD.

Kapitola 2

Základní pojmy z oblasti formálních jazyků

Jak bylo v úvodu řečeno, formální jazyky zastávají významné místo v oblasti modelování, popisu, analýze a v syntéze systémů. Tato kapitola je věnována základním pojmům oblasti formálních jazyků a některým základním operacím nad nimi, které budeme dále v textu využívat. Definice v této kapitole jsou převzaty z [6, 8].

2.1 Jazyky, základní pojmy a operace

Jako jedny z prvotních pojmů pro vymezení jazyka se setkáváme s pojmy abeceda a řetězec.

2.1.1 Abeceda, řetězec, operace nad řetězci

Definice 1. *Abeceda* je neprázdná množina prvků, které nazýváme symboly abecedy.

Příkladem abecedy může být $\Sigma = \{a, b\}$, kde abeceda Σ obsahuje dva symboly a a b.

Definice 2. *Řetězcem* (též větou nebo slovem) nad abecedou rozumíme posloupnost symbolů abecedy. Prázdnou posloupnost symbolů tj. posloupnost neobsahující žádný symbol, nazveme *prázdný řetězec* a označíme symbolem ε.
Nechť Σ je abecedou, pak:

1. prázdný řetězec ε je řetězcem nad abecedou Σ
2. je-li x řetězcem nad Σ a $a \in \Sigma$, pak xa je řetězcem nad Σ
3. y je řetězcem nad abecedou Σ právě tehdy, když jej lze získat aplikací pravidel 1 a 2

Budeme–li mít abecedu $\Sigma = \{a, b\}$, pak řetězec $w = aabaaa$ je řetězcem nad touto abecedou.

Definice 3. Nechť x a y jsou řetězce nad abecedou Σ. *Konkatenací řetězců* x a y vznikne řetězec xy připojením řetězce y za řetězec x. Takže pokud $x = x_1x_2 \cdots x_n$ a $y = y_1y_2 \cdots y_n$, pak konkatenace řetěžců x a y, značme xy, je řetězec

$$xy = x_1x_2 \cdots x_ny_1y_2 \cdots y_n$$

Následující definice vymezují další užitečné pojmy.

Definice 4. Nechť x je řetězcem nad abecedou Σ. Délku řetězce x, značme $|x|$, definujme takto:

- pokud $x = \varepsilon$, pak $|x| = 0$
- pokud $x = x_1 x_2 \cdots x_n$, pak $|x| = n$, pro $n \geq 1$ a $\forall i \in \mathbb{N}^+ : x_i \in \Sigma$

Definice 5. Nechť x je řetězcem nad abecedou Σ, pak x^R nazveme *reverzí řetězce* x, kde pro $x = \varepsilon$ platí, že $\varepsilon^R = \varepsilon$, a pokud $x = x_1 x_2 \cdots x_{n-1} x_n$, potom $x^R = x_n x_{n-1} \cdots x_2 x_1$.

Definice 6. Nechť w je řetězcem nad Σ. Řetězec z nazveme *podřetězcem* řetězce w, pokud existuje řetězec x a řetězec y takový, že $w = xzy$. Řetězec x_1 nazveme *prefixem* řetězce w, pokud existuje řetězec y_1 takový, že $w = x_1 y_1$. Řetězec y_2 nazveme *suffixem* řetězce w, pokud existuje řetězec x_2 takový, že $w = x_2 y_2$.

Definice 7. Nechť $x_1, x_2 \in \Sigma^*$ jsou dva řetězce nad abecedou Σ, a $a, b \in \Sigma$ jsou symboly. Paralelní kompozici definujeme rekurentně s využitím popisu ve tvaru regulárního výrazu:

$$ax_1 \parallel bx_2 = a(x_1 \parallel bx_2) + b(ax_1 \parallel x_2)$$

$$a \parallel \varepsilon = \varepsilon \parallel a = a$$

Poznamenejme, že výsledkem operace paralelní kompozice dvou řetězců je jazyk.

2.1.2 Jazyk a operace nad jazyky

Nyní, když máme potřebné informace o abecedách a řetězcích, můžeme přistoupit k vymezení pojmu jazyka.

Definice 8. Nechť Σ je abecedou. Symbolem Σ^* označíme množinu řetězců konečné délky nad abecedou Σ včetně prázdného řetězce. Symbolem Σ^+ množinu všech řetězců nad abecedou Σ vyjma prázdného řetězce ε, tj. $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$. Znakem Σ^ω označíme množinu řetězců nad abecedou Σ, kde pro řetězec $w \in \Sigma^\omega$ platí, že $|w| = \omega$. Množinu L, pro kterou platí $L \subseteq \Sigma^*$, případně $L \subseteq \Sigma^+$, nazveme *jazykem* L nad abecedou Σ. Množinu L^ω nazveme *ω–jazykem*, pokud $L^\omega \subseteq \Sigma^\omega$. Řetězec $w \in L$, nazveme *větou* (slovem) jazyka L.

Nad jazyky lze provádět i některé operace, se kterými se blíže seznámíme.

Definice 9. Nechť L_1 je jazykem nad abecedou Σ_1 a L_2 je jazykem nad abecedou Σ_2. Pak *konkatenací* (součinem) jazyků L_1 a L_2 je jazyk $L_1 \cdot L_2$ nad abecedou $\Sigma_{1,2} = \Sigma_1 \cup \Sigma_2$, definovaný takto:

$$L_1 \cdot L_2 = \{xy : x \in L_1 \wedge y \in L_2\}$$

Definice 10. Nechť Σ je abecedou a L_1, L_2 jazyky nad ní. Pak

- $L_1 \cup L_2 = \{x : x \in L_1 \vee x \in L_2\}$ nazveme *sjednocením jazyků* L_1 a L_2
- $L_1 \cap L_2 = \{x : x \in L_1 \wedge x \in L_2\}$ nazveme *průnikem jazyků* L_1 a L_2
- $L_1 \parallel L_2 = \{x \parallel y : x \in L_1 \wedge y \in L_2\}$ nazveme *paralelní kompozicí* jazyků L_1 a L_2

Definice 11. Nechť L je jazyk nad abecedou Σ. *Iteraci* L^* a *pozitivní iteraci* L^+ jazyka L definujeme takto:

$$L^0 = \varepsilon$$

$$L^n = L \cdot L^{n-1}, \text{ pro } n \geq 1$$

$$L^* = \bigcup_{n \geq 0} L^n$$

$$L^+ = \bigcup_{n \geq 1} L^n = L^* \setminus \{\varepsilon\}$$

Definice 12. Nechť L je jazykem nad abecedou Σ. *Reverzí jazyka* L^R rozumíme

$$L^R = \{w^R \ : \ \forall w \in L\}$$

Definice 13. Nechť Σ je abeceda, L jazyk nad abecedou Σ a Σ^* množina všech konečných řetězců. *Doplňkem jazyka* $\overline{L}$ rozumíme

$$\overline{L} = \{x : x \in \Sigma^* \wedge x \notin L\}, \text{ tj. } \overline{L} = \Sigma^* \setminus L$$

2.2 ω–jazyky

Jak bylo uvedeno, existuje třída tzv. *ω–jazyků*. Jedná se o třídu, která je významná pro popis systémů, které pracují s potenciálně nekonečnými vstupy nebo se u nich předpokládá neomezeně dlouhá doba běhu. Třída ω–jazyků je specifická tím, že všechna slova daného ω–jazyka mají nekonečnou délku, tj. $\forall w \in \Sigma^\omega \ : \ |w| = \omega$. ω–jazyky můžeme dělit například na základě schopnosti nějakého typu automatu příjout tento jazyk, jak bylo popsáno například v [11]. Na tomto základě se setkáváme s několika typy ω-jazyků. Můžeme uvést například ω–regulární jazyky (též regulární ω–jazyky) přijímané konečnými automaty (*Büchiho automaty*), bezkontextové ω–jazyky přijímané zásobníkovými automaty nebo ω–jazyky přijímané tzv. *Turingovými stroji*. V tomto textu, jelikož ω–jazyky nejsou jeho hlavní náplní, se omezíme pouze na popis ω–regulárních jazyků, které v současné době představují nejlépe prostudovanou třídu ω–jazyků. Uvedené definice jsou převzaty z [9].

Definice 14. Regulární množina nad konečnou abecedou Σ je definována jako

1. $\emptyset$ (prázdná množina), $\{\varepsilon\}$ (množina obsahující prázdný řetězec), $\{a\}$ pro všechna $a \in \Sigma$ jsou regulární množiny nad Σ
2. regulární množiny lze získat aplikací operací $\cup$, $\cdot$, * nad regulárními množinami

Abychom mohli popisovat ω–jazyky formou regulárních množin, je třeba definici regulárních množin rozšířit o definici nekonečné iterace $^\omega$ následovně

Definice 15. Nechť $\forall X \subset \Sigma^*$, kde Σ^* reprezentuje množinu konečných slov nad abecedou Σ, platí

$$X^\omega = \{x_0 x_1 \cdots : \forall i \geq 0, x_i \in X \setminus \{\varepsilon\}\}$$

Množina všech řetězců nad abecedou Σ je $X^\infty = X^* \cup X^\omega$.

Množina X^ω představuje množinu nekonečně dlouhých slov vzniklých konkatenací nekonečně mnoha neprázdných slov konečné délky obsažených v množině X. Pokud $X = \{u\}$ je $u = a_1a_2 \cdots a_n$ slovem množiny X, pak $X^\omega = \{u^\omega\}$, kde

$$u^\omega = a_1a_2 \cdots a_na_1a_2 \cdots a_na_1a_2 \cdots$$

je slovo nekonečné délky získané opakováním slova u nekonečně mnohokrát.

Když jsme si ukázali, jak lze získat základní množiny slov ω–jazyků prostřednictvím regulárních množin, můžeme přistoupit k definici pro tzv. ω–regulárních množin (v literatuře označované také jako ω–*rational subsets*).

Definice 16. Množina $\mathcal{R}$ je nejmenší množina regulárních podmnožin Σ^∞ taková, že

1. $\emptyset \in \mathcal{R}$, $\forall a \in \Sigma : \{a\} \in \mathcal{R}$
2. $\mathcal{R}$ je uzavřena vůči aplikaci konečného počtu operace sjednocení $\cup$
3. $\forall X \subseteq \Sigma^* \wedge \forall Y \subseteq \Sigma^\infty$, kde $X, Y \in \mathcal{R}$, pak $XY \in \mathcal{R}$
4. $\forall X \subseteq \Sigma^* : X \in \mathcal{R}, X^* \in \mathcal{R} : X^\omega \in \mathcal{R}$

Z hlediska ω–jazyků je zajímavá především ω-regulární podmnožina Σ^∞, která je zároveň obsažena i v Σ^ω. Tuto množinu nazveme jako ω–regulární podmnožinu Σ^ω.

Věta 1. Podmnožina Σ^ω je ω–regulární právě tehdy, když je konečným sjednocením množin ve formě XY^ω, kde X a Y jsou regulární podmnožiny Σ^*.

Množina řetězců vzniklá podle věty 1 je množinou řetězců ω–regulárního jazyka, které jsou rozeznatelné např. *Büchiho automaty.*
Z pohledu uzávěrových vlastností (viz [11, 9]) jsou ω–jazyky uzavřeny například k operacím:

- sjednocení ($\cup$) a průniku ($\cap$)
- nekonečná iteraci ($^\omega$)
- Levá konatenace, tj. pokud L_1 je ω–jazyk nad Σ^ω a L_2 je jazyk nad Σ^*, pak $L_2 \cdot L_1$ je také ω–jazykem
- Operaci $Pref(w)$, kde $Pref(w)$ je jazyk obsahující všechny konečné prefixy slova $w \in \Sigma^\omega$

Kapitola 3

Petriho sítě a jazyky Petriho sítí

Obsahem této kapitoly jsou P/T Petriho sítěmi a jejich úpravy, které umožňují specifikaci jazyků Petriho sítí. Tyto jazyky zařadíme do několika tříd, podle jejich vlastností a uvedeme si postavení jazyků Petriho sítí v rámci Chomského hierarchie. Závěr kapitoly bude věnován úvodu do vztahů Petriho sítí vůči ω–jazykům.

3.1 Petriho sítě

Petriho sítě vznikly jako zobecnení konečných automatů a jejich vyjadřovacích schopností. Umožňují tak modelovat např. paralelní a distribuované procesy (systémy) nebo kritické sekce řízení toku programu [10]. Petriho sítě podle použití na specifické účely dosáhly mnoha podob. V tomto textu se se setkáme s jednou z obecnějších forem Petriho sítí a to P/T Petriho sítěmi, někdy pro odlišení od tzv. *barvených Petriho sítí* nazývané jako *černobílé Petriho sítě*. V následujícím textu budeme pod pojmem Petriho síť rozumět P/T Petriho síť, seznámíme se s pojmy síť, Petriho síť, formami jejich zápisu a vlastnostmi Petriho sítí. Definice uvedené v podkapitolách jsou převzaty z [4, 5].

3.1.1 Petriho síť

Obecnou síť, kterou použijeme jako základ pro definici Petriho sítě, definujeme takto:

Definice 17. Trojice $N = (P, T, F)$ nazveme sítí, kde

1. P je množina míst
2. T je množina přechodů, kde $P \cap T = \emptyset$
3. $F \subseteq (P \times T) \cup (T \times P)$ je binární relací, kterou nazveme *tokovou relace* sítě N.

Jednoduchá síť je popsána na obrázku 3.1. Z obrázku je vidět, že síť v grafické podobě má formu bipartitního grafu, kde vrcholy grafu reprezentující místa mají tvar kruhu a uzly představující přechody tvar obdélníku. Relace mezi místy a přechody jsou znázorněny orientovanou hranou. Abychom mohli hovořit o Petriho sítích je nutné zavést některé základní prvky, které dodávají Petriho sítím jejich modelovací schopnosti. Pro zachycení tzv. stavu Petriho sítě zavedeme pojem značení Petriho sítě a s ním spojené pojmy kapacita místa, váha hrany a proveditelnost přechodu. Vlastní značení v grafu bude zachyceno značkami (tečkami, puntíky) v místech případně číslem. Nyní přistoupíme k vlastní definici Petriho sítě.

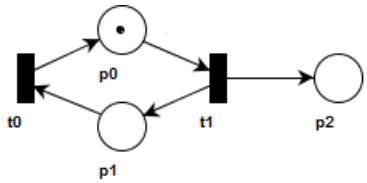
Obrázek 3.1: Jednoduchá síť

Definice 18. Nechť $PN = (P, T, F, W, K, M_0)$ je Petriho sítí, kde

1. trojice (P, T, F) reprezentuje konečnou síť jak jsme ji zavedli
2. $W : F \to \mathbf{N} \setminus \{0\}$ představuje *váhovou funkci*, která jednoznačně přiřadí váhu každé hraně sítě
3. $K : P \to \mathbf{N} \cup \{\omega\}$ reprezentuje *kapacitu míst*, tj. maximální počet značek, které je možné přiřadit každému místu $p \in P$
4. $M_0 : P \to \mathbf{N} \cup \{\omega\}$ nazveme *počátečním značením* míst sítě respektující kapacity míst, $\forall p \in P : M_0(p) \leq K(p)$

Konvence pro místa bez specifikované kapacity budeme předpokádat neomezenou kapacitu a pro hrany bez číselného ohodnocení budeme předpokládat váhu 1.

Jednoduchá Petriho síť je znázorněna na obrázku 3.2



Obrázek 3.2: Jednoduchá P/T Petriho síť

Definice 19. Značení Petriho sítě, popisující *stav Petriho sítě*, je jednoznačně určeno aktuálním rozmístěním značek v jednotlivých místech sítě.

Dalším krokem pro zavedení Petriho sítí je nutnost specifikovat, za jakých podmínek je přechod tzv. *proveditelný*. Provedením přechodu budeme rozumět změnu stavu Petriho sítě ze značení M do značení M'. Chování nebo též *výpočetní posloupnost* Petriho sítě můžeme rovněž chápat jako posloupnost změn značení při provádění jednotlivých přechodů $t \in T$.

Ještě než uvedeme pravidla pro provádění přechodů je vhodné zavést pojmy vstupní množina prvku (preset) a výstupní množina prvku (postset).

Definice 20. Nechť $N = (P, T, F)$ je síť

1. Pro všechna $x \in (P \cup T)$
   ${}^\bullet x = \{y : yFx\}$ nazveme *vstupní množinou* prvku x
   $x^\bullet = \{y : xFy\}$ nazveme *výstupní množinou* prvku x

2. Uspořádaná dvojice $\langle p, t\rangle \in (P \times T)$ se nazývá *vlastní cyklus*, jestliže $pFt \wedge tFp$. Pokud síť N takový cyklus neobsahuje nazveme ji *čistou sítí*.

Nyní můžeme přistoupit k vlastní definici pravidel pro provádění přechodů.

Definice 21. Nechť $PN = (P, T, F, W, K, M_0)$ je Petriho síť.

1. $M : P \to \mathbf{N} \cup \{\omega\}$ se nazývá značení Petriho sítě PN, jestliže $\forall p \in P : M_0(p) \leq K(p)$ Nechť M je značení Petriho sítě PN.

2. Přechod $t \in T$ je *proveditelný* při značení M (*M-proveditelný*), jestliže
$\forall p \in {}^\bullet t : M(p) \geq W(p, t)$
$\forall p \in t^\bullet : M(p) \leq K(p) - W(t, p)$

3. Je-li přechod $t \in T$ M-proveditelný, získáme *následné značení* M', které je definováno $\forall p \in P$ takto:
$$M'(p) = \begin{cases} M(p) - W(p,t), & \text{jestliže } p \in {}^\bullet t \setminus t^\bullet \\ M(p) + W(t,p), & \text{jestliže } p \in t^\bullet \setminus {}^\bullet t \\ M(p) - W(p,t) + W(t,p), & \text{jestliže } p \in {}^\bullet t \cap t^\bullet \\ M(p) & jinak \end{cases}$$
Provedení přechodu t ze značení M do značení M' symbolicky zapíšeme:
$$M[t\rangle M'$$

4. $[M\rangle$ označme nejmenší množinu různých značení sítě PN takovou, že
   (a) $M \in [M\rangle$
   (b) je-li $M_1 \in [M\rangle$ a pro nějaké $t \in T$ platí $M_1[t\rangle M_2$, pak $M_2 \in [M\rangle$

   Množina $[M\rangle$ se nazývá *množinou dosažitelných značení ze značení* M a množina $[M_0\rangle$ dosažitelných míst z počátečního značení se nazývá *množinou dosažitelných značení sítě PN*.

Na obrázku 3.3 je znázorněno provedení přechodu $t \in T$ a s ním spojená změna značení $M[t\rangle M'$.

Nyní se seznámíme s pojmem přechodová funkce a s jejím využití vyjádříme tzv. výpočetní posloupnost Petriho sítě.

Definice 22. Nechť $PN = (P, T, F, W, K, M_0)$ je Petriho síť a $[M_0\rangle$ reprezentuje množinu všech dosažitelných značení. *Přechodovou funkcí* Petriho sítě PN nazveme funkci δ

$$\delta : [M_0\rangle \times T \to [M_0\rangle, \text{ pro kterou}$$

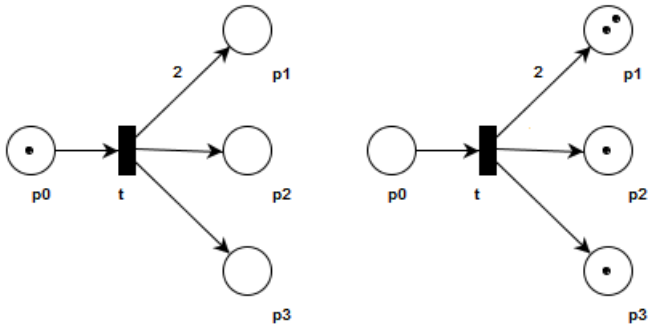
$$\forall t \in T : \forall M, M' \in [M_0\rangle : \delta(M, t) = M' \Leftrightarrow M[t\rangle M'$$

Přechodová funkce δ může být zobecněna na posloupnost přechodů

$$\delta : [M_0\rangle \times T^{**} \to [M_0\rangle$$

takto:

$$\delta(M, t\tau) = \delta(\delta(M, t), \tau), \tau \in T^{**}$$



Obrázek 3.3: Značení Petriho sítě před a po provedení přechodu t

$$\delta(M, \varepsilon) = M, \text{ kde } \varepsilon \text{ je prázdný symbol}$$

Posloupnost $\tau \in T^+$ nazveme *výpočetní posloupností* Petriho sítě PN, je-li pro ni definována přechodová funkce $\delta(M_0, \tau)$. Množina všech výpočetních posloupností charakterizuje chování Petriho sítě PN.

Výpočetní posloupnost Petriho sítě můžeme rovněž zapsat jako posloupnost změn značení při provádění přechodů:

$$M[t_1\rangle M_1[t_2\rangle \cdots [t_{n-1}\rangle M_{n-1}[t_n\rangle M'$$

3.1.2 Algebraická reprezentace Petriho sítí

Vedle grafické a formální reprezentace Petriho sítě existuje i algebraická reprezentace. Jedná se o popis Petriho sítě formou matic a vektorů. Blíže se seznámíme s pojmy toková matice, matice změn, vektor značení a jejich použití při provedení přechodu a získání nového vektoru značení.

Na přechod můžeme rovněž nahlížet jako na vektor, který nám určuje, jak se jeho provedením mění počet značek jednotlivých míst, což je vyjádřeno v definici 23.

Definice 23. Nechť $PN = (P, T, F, W, K, M_0)$ je Petriho síť.

1. Pro přechod $t \in T$ nechť je definován vektor $\underline{t} : P \to \mathbf{Z}$ takto:
   $\underline{t}(p) = W(t,p)$, jestliže $p \in t^\bullet \setminus {}^\bullet t$
   $\underline{t}(p) = -W(p,t)$, jestliže $p \in {}^\bullet t \setminus t^\bullet$
   $\underline{t}(p) = W(t,p) - W(p,t)$, jestliže $p \in t^\bullet \cap {}^\bullet t$
   $\underline{t}(p) = W(t,p)$, jinak
2. Matice Petriho sítě $\underline{N}$ je definována jako matice

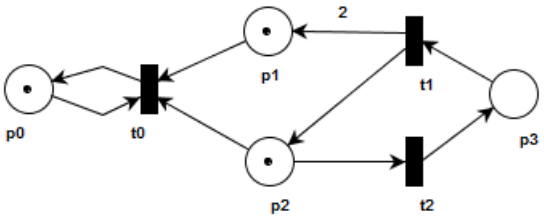
   $$\underline{N} : P \times T \to \mathbf{Z}, \text{ kde } \underline{N}(p,t) = \underline{t}(p)$$

   Tuto matici nazveme *matice změn Petriho sítě*.

S využitím vektorové reprezentace přechodu můžeme popsat, jakým způsobem se změní značení Petriho sítě při provedení přechodu $t \in T$.

Věta 2. Nechť $PN = (P, T, F, W, K, M_0)$ je Petriho síť a $M, M' : P \to \mathbf{N} \cup \{\infty\}$ jsou dvě značení Petriho sítě PN. Pak pro každý přechod $t \in T$, který je M-proveditelný platí:

$$M[t\rangle M' \Leftrightarrow M + \underline{t} = M'$$



Obrázek 3.4: Petriho síť a její matice

Matice změn $\underline{N}$ pro Petriho síť $PN = (P, T, F, W, K, M_0)$ na obrázku 3.4 s vekotrem počátečního značení sítě $M_0 = (1, 1, 1, 0)$.

$$\underline{N} = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{ccc} t_0 & t_1 & t_2 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ -1 & 2 & 0 \\ -1 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix} \end{array}$$

Alternativním maticovým vyjádřením je například tzv. *toková matice*. Tento typ matice je narozdíl od matice změn jednoznačný i pro sítě, které nejsou čisté tj. obsahují vlastní cyklus. Pro odlišení od matice změn budeme značit tokovou marici $\underline{F}$. Toková matice maticově reprezentuje váženou tokovou relaci F.

Definice 24. Nechť $PN = (P, T, F, W, K, M_0)$ je Petriho síť. Tokovou maticí sítě PN nazveme matici F, $F : P \times T \to \mathbf{N} \times \mathbf{N}$.

$$F(p,t) = \langle w(p,t), w(t,p)\rangle, \text{ kde } w(x,y) = \begin{cases} W(x,y), & \text{jestliže } \langle x,y\rangle \in F \\ 0, & \text{v opačném případě} \end{cases}$$

Pro Petriho síť $PN = (P, T, F, W, K, M_0)$ na obrázku 3.4 vypadá toková matice $\underline{F}$ takto:

$$\underline{F} = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{array} \begin{array}{c} \begin{array}{ccc} t_0 & t_1 & t_2 \end{array} \\ \begin{pmatrix} (1,1) & (0,0) & (0,0) \\ (1,0) & (0,2) & (0,0) \\ (1,0) & (0,1) & (1,0) \\ (0,0) & (1,0) & (0,1) \end{pmatrix} \end{array}$$

Někdy je toková matice $\underline{F}$ reprezentována dvěma celočíselnými maticemi $\underline{F}^+$ a $\underline{F}^-$.

$$\underline{F}^+ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \underline{F}^- = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

3.2 Jazyky Petriho sítí

Pro lepší možnosti specifikovat jazyky je vhodné rozšířit definici Petriho sítě, podobně jako u konečných automatů, o abecedu Petriho sítě a zavést mechanismus, který přiřadí symboly této abecedy jednotlivým přechodům sítě. O Petriho síti ohodnocené přechody budeme dále hovořit jako o ohodnocené Petriho síti. Představíme si některé třídy jazyků Petriho sítí a začleněníme je do Chomského hierarchie. Samostatná část je věnována vztahu Petriho sítí k ω–jazykům.

Z hlediska jazyků, je vhodné uvést, že dvě Petriho sítě jsou ekvivalentní, pokud přijímají stejný jazyk.

Pro zjednodušení zápisu Petriho sítě budeme předpokládat, že kapacita míst K bude neomezená a nebudeme ji v definici nadále uvádět, navíc ke každé Petriho síti s kapacitou lze nalézt metodou komplementace ekvivalentní Petriho síť takovou, že kapacita míst takto vzniklé Petriho sítě je neomezená.

3.2.1 Definice jazyků Petriho sítí

Zatím jsme popisovaly provádění přechodů pomocí jejich názvů, což nám znemožňuje modelovat například dva přechody, které reprezentují stejnou operaci nebo událost. Proto zavádíme *abecedu Petriho sítě*, značme ji Σ, a zobrazení λ, které přiřadí jednotlivým přechodům sítě symboly abecedy Σ. Zobrazení λ budeme nazývat *ohodnocení přechodů*. Definice jsou převzaty z [5, 4]. Z hlediska ohodnocení přechodů rozeznáváme tři základní typy:

1. sítě s injektivním ohodnocením $\lambda : T \to \Sigma$ (free-labeled Petri nets), kde

$$\lambda(t) = \lambda(t') \Rightarrow t = t'$$

2. ohodnocení $\lambda : T \to \Sigma$, připouštějící aby 2 přechody byly ohodnoceny stejným symbolem, nikoliv však prázdným symbolem ε

3. libovolné ohodnocení $\lambda : T \to \Sigma \cup \{\varepsilon\}$

Takto rozšířenou Petriho síť nazveme *ohodnocená Petriho síť*.

Definice 25. Nechť $PN = (P, T, F, W, M_0, \Sigma, \lambda)$ je ohodnocenou Petriho sítí, kde

- P, T, F, W, M_0 mají dosud užívaný význam
- Σ je abecedou Petriho sítě
- λ je ohodnocení přechodů Petriho sítě symboly abecedy Σ

V analogii k automatům zavedeme pojmy *počáteční místo* a *počáteční stav*. Počáteční stav odpovídá počátečnímu značení sítě M_0. Někdy je žádoucí, aby počáteční stav byl spojen s počátečním místem, značme p_s, tak aby platilo

$$M_0(p_s) = 1 \text{ a } \forall p \in P \setminus \{p_s\} : M_0(p) = 0$$

V podstatě je možné převést jakoukoliv síť $PN = (P, T, F, W, K, M_0)$ na ekvivalentní síť $PN' = (P', T', F', W', K', M_0')$ s počátečním místem následovně:

1. $P' = P \cup \{p_s\}$
2. $T' = T \cup \{t_s\}$
3. $F' = F \cup F_{t_s}$, kde $F_{t_s} = \{\langle p_s, t_s \rangle\} \cup \{\langle t_s, p \rangle\}$, pokud $M_0(p) \neq 0$
4. W' je rozšíření váhové funkce W:
   $W'(p_s, t_s) = 1 \wedge W'(t_s, p) = k \Leftrightarrow M_0(p) = k, k \in \mathbf{N} \setminus \{0\}$
5. K' je rozšířením $K : K'(p_s) = 1$
6. $M_0' : P \cup \{p_s\} \to \mathbf{N}, M_0'(p_s) = 1, \forall p \in P : M_0'(p) = 0$

Jinak řečeno na počátku je proveditelný pouze přechod t_s, jehož provedením získáme značení $M_0'(p) = M_0(p), \forall p \in P$. Množiny výpočetních posloupností sítí PN a PN' jsou shodné, pokud je $\lambda(t_s) = \varepsilon$ a $\forall t \in T : \lambda(t) = t$, jinak se liší pouze počátečním přechodem t_s.
Z hlediska koncových stavů rozlišujeme čtyři způsoby ukončení, na jejichž základě rozlišujeme i typy jazyků Petriho sítí. Jedná se o jazyky Petriho sítí **L**, **G**, **T**, **P**.

Definice 26. Nechť PN je Petriho síť, M_0 je počáteční značení PN, $\lambda : T \to \Sigma \cup \{\varepsilon\}$ je ohodnocení přechodů PN se zobecněnou přechodovou funkcí δ

$$\delta : [M_0\rangle \times T^* \to [M_0\rangle$$

a množinou koncových stavů $Q_f \subseteq [M_0\rangle$. Jazyk Petriho sítě $L(PN) \subseteq \Sigma^*$ je definovaný jako

$$L(PN) = \{\lambda(\alpha) : \alpha \in T^* \wedge \delta(M_0, \alpha) \in Q_f\}$$

se nazývá jazykem typu **L**.

Definice 27. Nechť PN je Petriho síť s počátečním značením M_0, ohodnocením přechodů λ a přechodovou funkcí δ a s konečnou množinou koncových stavů Q_f. Jazyk Petriho sítě $L(PN)$, pokud je definovaný jako

$$L(PN) = \{\lambda(\alpha) : \alpha \in T^* \wedge \exists M_f \in Q_f : \delta(M_0, \alpha) \geq M_f\}$$

nazveme jazykem typu **G**.

Definice 28. Nechť PN je Petriho síť s počátečním značením M_0, ohodnocením přechodů λ a přechodovou funkcí δ a s množinou koncových stavů Q_f. Jazyk Petriho sítě $L(PN)$ definovaný jako

$$L(PN) = \{\lambda(\alpha) : \alpha \in T^* \wedge \delta(M_0, \alpha) \in [M_0\rangle \delta(\delta(M_0, \alpha), t) \text{ je nedefinováno } \forall t \in T\}$$

nazveme jazykem typu **T**.

Definice 29. Nechť PN je Petriho síť s počátečním značením M_0, ohodnocením přechodů λ a přechodovou funkcí δ a s množinou koncových stavů Q_f. Jazyk Petriho sítě $L(PN)$ definovaný jako

$$L(PN) = \{\lambda(\alpha) : \alpha \in T^* \wedge \delta(M, \alpha) \in [M_0\rangle\}$$

nazveme jazykem typu **P**.
Jazyk typu **P** je tzv. *prefixovým jazykem.* Je-li $x \in L(PN)$ a $x = yz$, pak také $y \in L(PN)$.

	$\lambda : T \to \Sigma$ (injektivní zobrazení)	$\lambda : T \to \Sigma$ (bez ε–přechodů)	$\lambda : T \to \Sigma$ (s ε–přechody)
L typ	L^f	L	L^ε
G typ	G^f	G	G^ε
T typ	T^f	T	T^ε
P typ	P^f	P	P^ε

Tabulka 3.1: Tabulka tříd jazyků Petriho sítí podle způsobu ohodnocení přechodů

Třídy jazyků typu **L**, **G**, **T** a **P** lze na základě způsobu ohodnocení přechodů rozšířit na dvanáct tříd (viz tabulka 3.1). Pro třídy každého typu jazyka (L, G, T, P) platí, že

$$L^f \subseteq L \subseteq L^\varepsilon$$

$$G^f \subseteq G \subseteq G^\varepsilon$$

$$T^f \subseteq T \subseteq T^\varepsilon$$

$$P^f \subseteq P \subseteq P^\varepsilon$$

Mezi jednotlivými třídamy jazyků rovněž panují určité vztahy. Inkluze mezi jednotlivými třídamy, kde orientovaná hrana z A do B reprezentuje $A \supseteq B$, vypadají následovně

$$\begin{array}{ccccc} T^f & \leftarrow & T & \leftarrow & T^\varepsilon \\ & & \uparrow & & \updownarrow \\ L^f & \leftarrow & L & \leftarrow & L^\varepsilon \\ & & \downarrow & & \downarrow \\ G^f & \leftarrow & G & \leftarrow & G^\varepsilon \\ \downarrow & & \downarrow & & \downarrow \\ P^f & \leftarrow & P & \leftarrow & P^\varepsilon \end{array}$$

Další rozšíření základních tříd jazyků Petriho sítí bylo popsáno v [7] se zaměřením na koncová značení a uzavřenost vůči některým operacím (např. *). Vymezuje se zde několik nových tříd, z nichž některé jsou ekvivalentní některým ze základních tříd ($Fin = L$, $Triv = P$, $Id = G$) a jejich vzájemné vazby.

3.2.2 Vlastnosti jazyků Petriho sítí typu L

Pro názornost se omezíme na jazyky Petriho sítě typu L, která je dostatečně obecná a z hlediska vlastností nejvíce studována a analyzována. Uvedeme si uzávěrové vlastnosti a vymezíme si vztah jazyků Petriho sítí vzhledem k Chomského hierarchii.

Standardní tvar ohodnocené Petriho sítě

Pro lepší názornost je vhodné zavést tzv. *standardní tvar Petriho sítě*, který umožňuje více se přiblížit způsob reprezentace sítě s počátečním místem a množinou koncových stavů podobně jako je tomu u konečných automatů.

Definice 30. Nechť $PN = (P, T, F, W, M_0, p_s, \Sigma, \lambda, P_f, Q_f)$ je ohodnocenou Petriho sítí ve standardním tvaru, kde

1. P, T, F, W, M_0, Q_f, Σ mají doposud používaný význam

2. $p_s \in P$ reprezentuje počáteční místo
   (a) $M_0(p_s) = 1 \wedge \forall p \in P \setminus \{p_s\} : M_0(p) = 0$
   (b) $\forall t \in T : \langle t, p_s \rangle \notin F$
3. $\lambda : T \to \Sigma$ je ohodnocení přechodů sítě
4. (a) $P_f \subseteq P, P_f = \begin{cases} \{p_f, p_s\} & \text{, jestliže } \varepsilon \in L(PN) \\ \{p_f\} & \text{, jestliže } \varepsilon \notin L(PN) \end{cases}$
   je množina koncových míst
   (b) $\forall t \in T : \langle p_f, t \rangle \notin F$
   (c) Je-li $M(p_t) \geq 0$ pro nějaké $M \in [M_0\rangle$, pak $\delta(M, t)$ je definována $\forall t \in T$

Věta 3. Ke každé ohodnocené Petriho síti PN typu L lze zkonstruovat ekvivalentní ohodnocenou Petriho síť ve standardním tvaru PN', tj. $L(PN) = L(PN')$.

(viz literatura [4, 5])

Z hlediska uzávěrových vlastností jsou jazyky Petriho sítí uzavřeny vzhledem k operacím konkatenace, sjednocení, průnik, paraelní kompozice,reverze jazyka v libovolném pořadí a množství opakování, vzhledem k iteraci jazyka jsou uzavřeny tzv. *řádně ukončující* Petriho sítě, tj. sítě u nichž je po ukončení provádění výpočetní posloupnosti jedna značka v koncovém místě a počet značek vytvořených v průběhu provádění výpočetní posloupnosti je konečný. Tato třída je ekvivalentní nějakému konečenému automatu a je třídou regulárních jazyků. Bližší informace v literatuře [5, 4].

Vztah jazyků Petriho sítí k Chomského hierarchii

Dalším krokem k vymezení vyjadřovacích schopností Petriho sítí je zařazení jazyků Petriho sítí v rámci Chomského hierarchie.

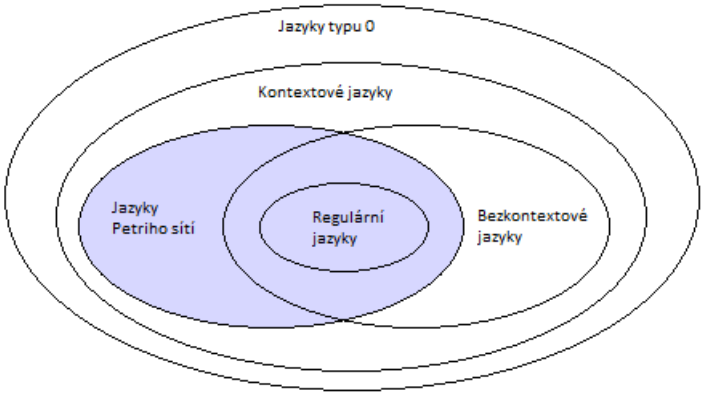
Každý regulární jazyk je jazykem generovaným Petriho sítí. Problematikou Petriho sítí a regulárních jazyků se zabýval například Valk [13]. Ke každému konečnému automatu lze sestrojit ohodnocenou Petriho síť, což ilustruje obrázek 3.5. Jazyk $L = \{ww^R : w \in \Sigma^*\}$ není

Obrázek 3.5: Převod konečného automatu na ohodnocenou Petriho síť

jazykem Petriho sítí. Třída jazyků Petriho sítí a třída bezkontextových jazyků je nesrovnatelná vzhledem k inkluzi. Všechny jazyky generované Petriho sítěmi jsou jazyky kontextové. Vlastí vztah jazyků Petriho sítí v rámci Chomského hierarchie je uveden na obrázku 3.6

3.2.3 Jazyky Petriho sítí a ω–jazyky

Podobně jako u *Büchiho automatů* je snaha i u Petriho sítí nalézt způsob, jakým mohou specifikovat ω–jazyky. Hledají se takové výpočetní posloupnosti, které dosahují opakovaně



Obrázek 3.6: Jazyky Petriho sítí v rámci Chomského hierarchie

značení $M \in [M\rangle$. V literatuře [14] se setkáváme s několika základními typy ω–jazyků, které jsou přijmány ohodnocenými Petriho sítěmi. Pro následující úsek textu budeme předpokládat ohodnocenou Petriho síť $PN = (P, T, F, W, M_0, \Sigma, \lambda)$ tak, jak jsme si ji zavedli. Zaveďme notaci $M[\alpha\rangle = M_0[t_0\rangle M_1[t_1\rangle \cdots$, kde α reprezentuje posloupnost přechodů, $\alpha(n)$ přechod na pozici n, kde $n \geq 0$ a $\lambda(\alpha) = \lambda(\alpha(0))\lambda(\alpha(1)) \cdots = \lambda(t_0)\lambda(t_1) \cdots$ představuje ohodnocení symboly této abecedy. $\uparrow Q_f$ představuje množinu značení takových, že

$$\uparrow Q_f = \{M', M \in [M\rangle : M' \geq M \wedge M \in Q_f\},$$

což je určitou analogií k jazykům typu G. Na tomto základě definujeme tři základní typy ω–jazyků specifikovaných Petriho sítěmi, kde α reprezentuje nekonečně dlouhou výpočetní posloupnost Petriho sítě PN a $\lambda(\alpha) \in \Sigma^\omega$ je slovo ω–jazyka. Tyto typy jsou:

$$L_{True}(PN) = \{\lambda(\alpha)\}$$

$$L_{ran\sqcap}(PN) = \{\lambda(\alpha) : \alpha \text{ dosáhne značení } \uparrow Q_f\}$$

$$L_{inf\sqcap}(PN) = \{\lambda(\alpha) : \alpha \text{ dosáhne značení } \uparrow Q_f \text{ nekonečně mnohokrát}\}$$

Třídu ω–jazyků $\mathbf{P}_\gamma$, kde $\gamma \in \{True, ran\sqcap, inf\sqcap\}$, symbolicky zapíšeme $L_\gamma(PN)$. Třída $L_\gamma(PN)$ koresponduje k třídě jazyků definované konečným automatem s obdobnou ukončující podmínkou.

Předešlé tři třídy jazyků jsou dále zkoumány a rozšiřovány o podtřídy. Pro $\alpha \in \Sigma^\omega$ specifikujeme operace *inf*, *ran* takto:

$$ran(\alpha) = \{a : a = \alpha(n) \text{ pro nějaké } n\}$$

$$inf(\alpha) = \{a : a = \alpha(n) \text{ pro nekonečně mnoho } n\}$$

tj. $ran(\alpha)$ je množinou symbolů vyskytujících se v řetězci α v konečném počtu a $inf(\alpha)$ je množina symbolů vyskytující se v řetězci α nekonečně mnohokrát.

Pro Petriho síť $PN = (P, T, F, W, M_0, \Sigma, \lambda)$ a $\alpha \in T^\omega$ zavedeme $PN(\alpha) = M_0 M_1 M_2 \cdots$,

pokud $M_0[\alpha(0)\rangle M_1[\alpha(1)\rangle M_2 \cdots$. Na tomto základě získáváme podtřídy ω–jazyků přijímaných Petriho sítí PN a to:

$$L_{True}(PN) = \{\lambda(\alpha) : M_0[\alpha\rangle\}$$

$$L_{ran\sqcap}(PN) = \{\lambda(\alpha) : M_0[\alpha\rangle \wedge ran(PN(\alpha)) \cap \uparrow Q_f \neq \emptyset)$$

$$L_{ran\subseteq}(PN) = \{\lambda(\alpha) : M_0[\alpha\rangle \wedge ran(PN(\alpha)) \subseteq \uparrow Q_f\}$$

$$L_{inf\sqcap}(PN) = \{\lambda(\alpha) : M_0[\alpha\rangle \wedge inf(PN(\alpha)) \cap \uparrow Q_f \neq \emptyset\}$$

$$L_{inf\subseteq}(PN) = \{\lambda(\alpha) : M_0[\alpha\rangle \wedge inf(PN(\alpha)) \subseteq \uparrow Q_f\}$$

Podle teorému **Theorem 4** uvedeneho v [14]

$$P_{True} = P_{ran\subseteq} \subset P_{ran\sqcap} = P_{inf\subseteq} \subset P_{inf\sqcap}$$

vycházejícího z topologické charakteristiky pro ω–regulární jazyky, můžeme nalézt tyto příklady jazyků:

$$a^*b^\omega \in P_{ran\sqcap} \setminus P_{True}$$

$$(a^*b)^\omega \in P_{inf\sqcap} \setminus P_{ran\sqcap}$$

ω–regulární jazyky, které byly popsány v sekci 2.2, lze specifikovat pomocí Petriho sítí. Vezmeme-li do úvahy, že pro každý konečný automat lze zkonstruovat ekvivalentní ohodnocenou Petriho síť (viz obrázek 3.5), můžeme uvažovat i o sítích specifikující ekvivalentní jazyk jako nějaký Büchiho automat, kde převod Büchiho automatu na Petriho síť je analogický jako pro klasické automaty.

Kapitola 4

Návrh akceptoru a jeho realizace

Jádrem této práce je implementace akceptoru jazyků Petriho sítí v programovacám jazyce C/C++ a je funkčí na počítačových stanicích s operačním systémem na bázi Unixu, který bude případně využitelný pro výukové účely. Díky využití standardních knihoven je akceptor spustitelný i na stanicích s operačním systémem MS Windows (testováno pro Windows 7 a 8).

Seznámíme se s problematikou návrhu akceptoru a způsobem realizace jeho implementace, dále pak s problémy, které návrh provázely, a jejich řešení v rámci návrhu a implementace.

4.1 Akceptor jazyku Petriho sítě a jeho programová realizace

Ještě před vlastní implementací akceptoru bylo třeba řešit několik otázek týkajících se zejména formy zadávání Petriho sítě, abecedy, testování přijetí jazyka podle typu třídy jazyka, dále pak vnitřní způsob reprezentace sítě. Rovněž bylo vhodné implementovat i ruční zadávání řetězců, kde by bylo možné vybírat z proveditelných přechodů v případě, že by byly dva a více přechody ohodnoceny stejným symbolem abecedy a zároveň byly proveditelné. V neposlední řadě Petriho sítě mohou sloužit i jako generátor řetězců příslušné třídy jazyka, proto je vhodné rozšířit akceptor o funkci generátoru řetězců jazyka příslušné třídy Petriho sítě.

Jak bylo zmíněno v úvodu, Petriho sítě se vytváři zejména pomocí grafických editorů. Protože akceptor je implementován formou konzolové aplikace a neobsahuje vlastní grafický editor, bylo vhodné využít některého ze existujících editorů v našem případě *PIPE 2* [2].

PIPE 2 ("Platform Independent Petri net Editor 2") představuje multiplatformní nástroj pro tvorbu a analýzu Petriho sítí napsaný v programovacím jazyce Java. Umožňuje uchovávat síť v XML souboru a její grafickou reprezentaci ukládat například ve fromátech png a PostSkript, což umožňuje jejich další využití.

Petriho síť zadaná pomocí nástroje *PIPE 2* je možné uložit ve formátu XML[1], který v dnešní době představuje jeden z nejvíce rozšířených formátu pro výměnu dat mezi aplikacemi. V souvislosti s využitím formátu XML bylo třeba řešit způsob, jakým se bude s tímto

[1] standardizovaný značkovací jazyk určený především pro přenosdat mezi aplikacemi

formátem pracovat. Opět bylo vhodné využít některý z existujících nástrojů, v našem případě se jedná o *CMarkup*[2][1], což je knihovna určená pro snadnou práci s XML soubory. Z XML souboru, který reprezentuje Petriho síť, lze získat informace o pojmenování míst, přechodů, počátečním značení a relacích mezi místy a přechody včetně vah hran. Akceptor nepředpokládá kapacitu místa, proto je implicitně brána jako nekonečná.

Další problém byl, jakým způsobem řešit načítání vstupní abecedy sítě. Při využití *PIPE 2* se vyskytla komplikace při ukládání sítí, které mají stejně pojmenované prvky (přechody). Tuto situaci bylo třeba řešit formou zavedení speciální notace popisu přechodů v názvech přechodů. Byla zvolena notace:

$$t(a),$$

kde a je symbolem abecedy Petriho sítě Σ a je jím ohodnocen přechod $t \in T$, tj. $\lambda(t) = t(a)$.

Pro reprezentaci Petriho sítě v rámci aplikace bylo vhodné využít algebraickou reprezentaci ve formě matice změn resp. tokových matic $\underline{F}^+$ a $\underline{F}^-$.

Z hlediska definování koncových značení Petriho sítě bylo nutné nalézt vhodý formát a způsob reprezentace, což je řešeno zavedením dalšího vstupního souboru s koncovými značeními Q_f. Jednotlivá značení jsou zapsána ve formátu vektoru značení a jsou oddělená středníkem např. $(0, 1, 1, 0)$;.

Jelikož se jedná o konzolovou aplikaci, bylo nutné řešit i jak vhodně zobrazit jednotlivé výpočetní posloupnosti. Což je opět řešeno formou zápisu změny značení při provedení přechodu

$$M[t\rangle M'$$

formou vektorového zapisu např.:

$$(0, 1, 0, 0)[t(a)\rangle(0, 1, 1, 0)$$

Oproti klasickému zápisu je zde navíc uveden symbol abecedy, kterým je tento přechod ohodnocen. Akceptor lze popsat s využitím modifikované definice ohodnocené Petriho sítě ve standardním tvaru:
Nechť $PN = (P, T, F, W, M_0, \Sigma, \lambda, P_f, Q_f)$ je ohodnocenou Petriho sítí ve standardním tvaru bez počátečního místa p_s, pak

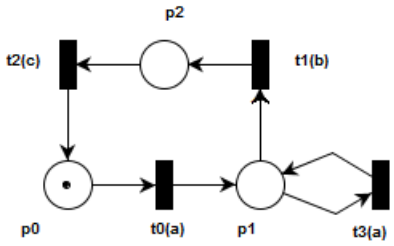
1. P je množina míst
2. T je množina přechodů
3. F je toková relace mezi místy a přechody a W je váha hrany (relace), F a W jsou reprezentovány formou matice změn, resp. tokových matic $\underline{F}^+$ a $\underline{F}^-$
4. M_0 je počáteční značení sítě
5. Σ je abeceda Petriho sítě, jejíž symboly jsou uvedeny v názvech přechodů sítě
6. λ je ohodnocením přechodů symboly abecedy Σ, kdy ohodncoení přechodů je reprezentováno v názvech jednotlivých přechodů pomocí notace $t(a), a \in \Sigma, t \in T$

[2]dostupné na adrese `<http://www.firstobject.com/dn_markup.htm>`, volně šiřitelný pro nekomerční použití

7. Q_f a P_f jsou reprezentovány souborem s koncovými značeními sítě, které jsou zapsané formou vektorů znační

Koncová značení mají význam pouze pro jazyky Petriho sítí typu L a G. Jazyky Petriho sítí typu T a P mají jiné mechanismy pro vyhodnocení koncových podmínek (viz 3.2).

Další úskalí z hlediska využití sítě jako generátoru jazyka přinesly sítě, které nejsou čisté, nebo obsahují takovou výpočetní posloupnost, ve které se dostáváme do značení, které je v porovnání s některým z předchozích značení stejné nebo toto značení pokrývá. Příklad takové sítě je na obrázku 4.1. Tyto sítě jsou potenciálně schopné generovat i neomezeně dlouhé řetězce. Tento problém řeší zavedení maximální délky řetězce, který bude Petriho sítí generován a zvolení vhodné meximální délky řetězce je ponecháno na uživateli.



Obrázek 4.1: Ukázka Petriho sítě generující potenciálně neomezeně dlouhé řetězce

4.2 Algoritmy a řešení implementace

Doposud jsme hovořili o návrhu vlastního akceptoru. Nyní se zaměříme na způsob, jakým je tento návrh reprezentován formou algoritmů a blíže se seznámíme s některými implementačními detaily, které bylo třeba řešit. Seznámíme se s obecně zapsanými algoritmy použitými pro rozhodování o přijetí nebo nepřijetí řetězce, pro generování řetězců jazyka a pro manuální vstup.

4.2.1 Načtení a zpracování vstupních dat

Pro vhodnou vnitřní reprezentaci sítě a případných koncových značení je nutné, aby byla vstupní síť správně zadána. Proto je nutné zkoumat formát sítě, zejména z pohledu správného naplnění příslušných matic a vekotrů. Také je nutné zajistit, aby přechody sítě byly řádně ohodnoceny právě jedním znakem abecedy sítě Σ. Vlastní abeceda Petriho sítě je vložena formou ohodnocení přechodů sítě. Dále je třeba se zaměřit na zadávání koncových značení pro příslušné jazyky Petriho sítě (typ L a G). Pokud si uživatel přeje vyjádřit (platí zejména pro jazyky typu L), že všechna možná značení daného místa $p \in P$ jsou přípustná v rámci vektoru značení jako koncové značení tohoto místa, ale ostatní místa mají koncové značení specifické, je mu dána možnost uvést do zápisu koncových značení ohodnocení místa symbolem X nebo x například:

$$(0, 1, X, 0);$$

Je předpokládáno, že soubor obsahuje právě jedno koncové značení v přislušném formátu na řádek.

Vlastní získání informací o síti probíhá v rámci načítání XML souboru. Tento soubor obsahuje informace o pojmenování míst a jejich počátečním značení, pojmenování přechodů, z něhož je třeba získat ohodnocení přechodu symbolem abecedy a informace o orientaci hran sítě a jejich vahách. Na základě těchto informací se sestavují matice změn resp. tokové matice $\underline{F}^+$ a $\underline{F}^-$, čímž se ve spojení s načteným vektorem počátečních značení a vektorem ohodnocení přechodů inicializuje vnitřní reprezentace Petriho sítě.

4.2.2 Rozhodnutí o přijetí řetězce jazyka

V rámci automatického rozhodnutí o přijetí nebo nepřijetí vstupního řetězce Petriho sítí je třeba zohlednit i typ sítě, pro který je daný řetězec určen. Z toho plyne nutnost testovat řetěze k příslušenému typu jazyka Petriho sítě, což je znázorněno v algorimu 1.

Algoritmus 1 Rozlišení typu jazyka

Předpokládá se načtená síť $PN = (P, T, F, W, M_0, \Sigma, \lambda, R)$, kde R reprezentuje množinu koncových značení.

function RUNENDTEST(M, typ jazyka *type*, množina koncových značení R)
- **if** $type = L$ **then**
  - **if** $M \in R$ **then**
    - **return** True
  - **else**
    - **return** False
  - **end if**
- **else if** $type = G$ **then**
  - **for** $r \in R$ **do**
    - **if** $\forall p \in P : M(p) \geq r(p)$ **then**
      - **return** True
    - **end if**
  - **end for**
  - **return** False
- **else if** $type = T$ **then**
  - **if** $\forall t \in T : t$ je neproveditelný při značení M **then**
    - **return** True
  - **else**
    - **return** False
  - **end if**
- **else if** $type = P$ **then**
  - **return** True
- **else**
  - **return** False
- **end if**

end function

Vlastní proces přijímání řetězce představuje algoritmus 2.

Algoritmus 2 Automatické vyhodnocení přijetí řetězce

Předpokládá se načtená síť $PN = (P, T, F, W, M_0, \Sigma, \lambda, R)$, kde R reprezentuje množinu koncových značení.

function FINDNEXTFIREABLETRANSSEQ(vstupní řetězec w, *index* hledaného symbolu v retezci w, aktuální značení M)
  if řetězec přečten **then**
    return RUNENDTEST(M, typ jazyka *type*, množina koncových značení R)
  end if
  if Najdi první netestovaný přechod t ohodnocený symbolem $w(index)$ **then**
    if nenalezen **then**
      return False
    end if
  end if
  if Není-li t proveditelný **then**
    return FINDNEXTFIREABLETRANSSEQ(w, *index*, M)
  else
    spočítej nové značení M' vzniklé provedením přechodu t, $M[t\rangle M'$
    ulož změnu značení sítě
    $index = index + 1$ ▷ *index* na další symbol vstupního řetězce w
    return FINDNEXTFIREABLETRANSSEQ(w, *index*, M')
  end if
end function

Algoritmus 2 vyhledává první možnou výpočetní posloupnost takovou, že celý vstupní řetězec je přijat a zároveň je splněna podmínka, že řetězc je řetězcem jazyka Petriho sítě. V případě nesplnění podmínky nebo nemožnosti přijmout aktuální symbol dochází k návratu a pokračování v hledání dalšího proveditelného přechodu ohodnoceného daným symbolem, případně se vrací o symbol (resp. symboly), což umožněno rekurzivním voláním této funkce.

4.2.3 Manuální zadávání řetězce jazyka

Součástí rozšíření implmentace automatizovaného akceptoru je možnost manuálně zadávat symboly nebo sekvence symbolů a v případě nalezení více proveditelných přechodů z aktuálního značení umožnit uživateli možnost rozhodnout se, který z přechodů má být proveden. Z důvodu výukových nebo i možnosti chybného zadání přechodu je vhodné, aby měl možnost navracení. Aby nebylo nutné znovu spouštět celou aplikaci se stejnými parametry, mělo by být umožněno opětovné načtení počátečních hodnot sítě (počáteční značení, vymazání přečtené části řetězce, atd.). Z předcházejících důvodů bylo zavedeno několik příkazů pro ovádání, viz tabulka 4.1.

Aby byl výběr přechodu jednoznačný, jsou vypsány pouze přechody proveditelné z aktuálního značení ohodnocené zadaným symbolem. Tyto přechody jsou následně očíslovány a popsány, tak jak byly zadány v rámci Petriho sítě vyrvořené editorem *PIPE 2*. Řádek tohoto výpisu může vypadat například takto:

```
1) t1(a)
```

–	návrat o jeden znak
exit	ukončení manuálního vstupu a rozhodnutí o přijetí zadaného řetězce požadovaným typem sítě
clear	uvedení sítě do stavu před začátkem přijímání řetězců (doposud zadaný řetězec je smazán)

Tabulka 4.1: Příkazy pro ovládání manuálního zadávání

Program očekává zadání příslušného čísla. V případě, kdy je na vstupu řetězec, je tento řetězec čten, dokud je provádění přechodů jednoznačné. Po ukončení manuálního zadávání symbolů příkazem `exit` dojde k otestování celé vstupní posloupností symbolů a rozhodnutí, zda vstupní řetězcem je řetězcem jazyka Petriho sítě.

4.2.4 Generování řetězců jazyka

Generování řetězců příslušného typu jazyka přináší již zmiňované úskalí v podobě počtu vytvořených řetězců a v délce těchto řetězců. Z tohoto důvoju je zavedeno omezení maximální délky řetězce zadané při spuštění programu. Nevhodně zvolená délka může vést k dlouhé době výpočtu nebo i k vyčerpání paměti počítače, protože u některých sítí může být počet prohledávaných stavů a testovaných řetězců neomezený. Problematické jsou sítě obsahující například vlastní cyklus nebo dosajují ve své výpočetní posloupnosti opakovaně nějakého značení. Funkci generátoru popisuje algoritmus 3.

Algoritmus 3 Generování řetězců

Předpokládá se načtená síť $PN = (P, T, F, W, M_0, \Sigma, \lambda, R)$, kde R reprezentuje množinu koncových značení.

```
function GENERATE(M, prefix, max, množina koncových značení R, typ jazyka type)
    if délka prefixu > max then
        return
    end if
    for všechny přechody t ∈ T do
        if pokud je přechod t proveditelný then
            spočítej značení M' vzniklé provedením t, M[t⟩M'
            prefix = prefix + λ(t)
            if max = délce prefixu then
                if RUNENDTEST(M', type, R) then
                    ulož změnu značení sítě M' a s ní spojený prefix
                end if
            end if
            GENERATE(M', prefix, max, R, type)
        end if
    end for
end function
```

Algoritmus 3 opět zohledňuje typ jazyka sítě a uchovává pouze řetězce, které jsou daným typem jazyka přijímány. Algoritmus je implementován jako algoritmus rekurzivního prohledávání do hloubky se zarážkou.

4.3 Spuštění aplikace

Jelikož se jedná o konzolovou aplikaci a je nutné již při vstupu zadávat některé informace, které jsou potřebné pro její správný běh. Mezi tyto informace patří například název vstupního souboru se sítí, soubor koncových značení, typ jazyka a další. V této části si uvedeme přehled přepínačů, které je možné zadat.

- `-h` – výpis nápovědy
- `-t` – výpis tokové matice a matice změn
- `-p` – zapnutí tisku výsledné výpočetní posloupnosti
- `-f <soubor se sítí>` – specifikuje název a umístění vstupního souboru s Petriho sítí podle řetězce v argumentu `<soubor>`
- `-r <soubor>` – specifikuje název a umístění vstupního souboru koncových značení podle řetězce v argumentu `<soubor>`
- `-m` – vytiskne koncová značení ve formátu stejném jako je ve vstupním souboru koncových značení
- `-s <řetězec>` – úvodní řetězec pro přečtení nebo prefix pro ruční vstup, řetězec uveden jako argument *řetězec*
- `-l <typ jazyka>` – `<typ jazyka>` určuje třídu jazyka Petriho sítě, pro kterou je akceptor spuštěn (viz tabulka 4.3)

Typ	**argument přepínače**
L	`l` nebo `L`
G	`g` nebo `G`
T	`t` nebo `T`
P	`p` nebo `P`

Tabulka 4.2: Typ jazyka na základě argumentu přepínače `-l`

- `-i` – spuštění ručního vstupu lze kombinovat s přepínačem `-s` pro zadání prefixové části řetězce
- `-g <délka řetězce>` – spuštění generátoru řetězců pro třídu jazyka určenou přepínačem `-r`, argument přepínače `<délka řetězce>` určuje celočíselnou hodnotu délky genererovaných řetězců větší než 0
- `-a` – modifikátor upravující chování přepínače `-g <délka řetězce>` tak, aby tisknul všechny přijaté řetězce do délky určené argumentem přepínače `<délka řetězce>`
- `-o` – modifikátor upravující chování přepínače `-g <délka řetězce>` tak, aby náhodně vybral jeden přijatý řetězec o délce určené argumentem přepínače `<délka řetězce>`

V případě neudání jednoho z přepínačů `-i` a `-g` se předpokládá zadání vstupního řetězce pro automatické vyhodnocení přijetí řetězce. Přepínač `-r <soubor>` je třeba uvádět pouze pro sítě specifikující jazyky typu L a G. Přepínač `-s` lze kombinovat s přepínačem `-i` a je

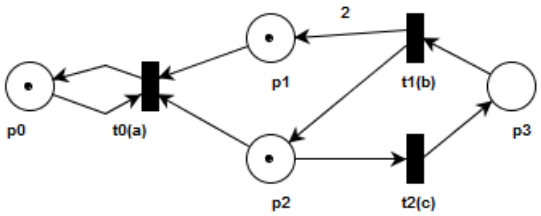
vyžadován v případě neuvedení přepínačů `-i` a `-g`. Přepínač `-p` má smysl zapínat pouze pokud není uveden ani jeden z přepínačů `-i` a `-g`, pro přepínač `-i` se volá tisk výpočetní posloupnosti po každém vstupu. Přepínače `-f <souboru se sítí>` a `-l <typ jazyka>` jsou oba povinné. Přepínače `-a` a `-o` lze zadat pouze v kombinaci s přepínačem `-g <délka řetězce>`. V případě zadání přepínače `-h` jsou ostatní přepínače ignorovány. Přepínač *-t* lze kombinovat se všemi typy běhu akceptoru a umožňuje výpis matic pro specifikovanou síť bez zadání vstupního řetězce k analýze.

Kapitola 5

Ověření funkčnosti návrhu a implementace

Součástí práce je rovněž ověřit vlastní funkčnost implementace akceptoru tak, aby na zadaný vstup vracel odpovídající výstup. Proto bylo vhodné najít takové příklady, kde by vstupy a výstupy byly známé a ověřené a na nich otestovat, zda na stejný vstup dostaneme očekávaný výstup.

Základem pro správnou funkčnost akceptoru je korektní načtení vstupních dat o síti. Jako příklad můžeme použít síť na obrázku 3.4 uvedenou v sekci 3.1.2, kterou navíc ohodnotíme symboly abecedy $\Sigma = \{a, b, c\}$ jak je znázorněno na obrázku 5.1.

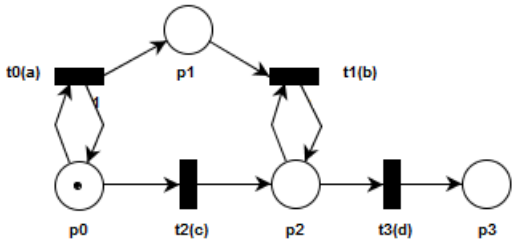


Obrázek 5.1: Petriho síť použitá pro testování načtení vstupních dat

5.1 Test vnitřní reprezentace sítě

Po spuštění aplikace s parametry `./bp -f test/test_matrix.xml -t -p` na výstupu získáváme tokovou matici a matici změn tak, jak byly popsané v kapitole 3.1.2, s odpovídající abecedou Σ (ve výstupu uvedeno jako `A`) a výpočetní posloupností v nultém kroku posloupnosti, která odpovídá počátečnímu značení sítě $M_0 = (1, 1, 1, 0)$ viz příloha B.1. Příkladem sítě, jejíž vstupy i výstupy jsou známé může být například síť na obrázku 5.2 [1].

[1] Síť převzata z `<http://www.fit.vutbr.cz/study/courses/PES/public/Prednasky/PES-07-Jazyky.pdf>` strana 12



Obrázek 5.2: Ohodnocená Petriho síť použitá pro testování výstupu

5.2 Testování generátoru a automatického přijímání řetězce

Pro Petriho síť PN na obrázku 5.2 je předpokládané počáteční značení $M_0 = (1, 0, 0, 0)$, množina koncových stavů $Q_f = \{(0, 0, 1, 0)\}$ a očekávané výstupy jsou:

1. L–typ: $L(PN) = \{a^n cb^n : n \geq 0\}$
2. G–typ: $L(PN) = \{a^m cb^n : m \geq n \geq 0\}$
3. T–typ: $L(PN) = \{a^m cb^n d : m \geq n \geq 0\}$
4. P–typ: $L(PN) = \{a^m : m \geq 0\} \cup \{a^m cb^n : m \geq n \geq 0\} \cup \{a^m cb^n d : m \geq n \geq 0\}$

Vlastní testování aplikace na očekávané výstupy lze rozdělit na testy pro generátor, pro automatické vyhodnocení přijetí zadaného řetězce a pro manuální zadávání řetězce zvoleného typu jazyka. Nejdříve otestujeme, zda aplikace se zadanou sítí na obrázku 5.2 generuje příslušné řetězce pro daný typ jazyka, poté otestujeme automatické přijímání některého z takto vygenerovaných řetězců. Příkazy použité pro generování řetězců jazyka daného typu jsou tyto:

1. L–typ: `./bp -l l -r rule.txt -f test/sit_test_jazyk.xml -g 10 -a`
2. G–typ: `./bp -l g -r rule.txt -f test/sit_test_jazyk.xml -g 10 -a`
3. T–typ: `./bp -l t -f test/sit_test_jazyk.xml -g 10 -a`
4. P–typ: `./bp -l p -f test/sit_test_jazyk.xml -g 10 -a`

Pro jednotlivé typy jazyka Petriho sítě získáváme řetězce (doplněno o informaci o celkovém počtu řetězců) jako jsou:

1. L–typ: `aaaacbbbb`, `aaacbbb`, celkový počet řetězců do délky řetězce 10 symbolů: 5
2. G–typ: `aaaaaaaacb`, `aaaaacbbb`, celkový počet řetězců do délky řetězce 10 symbolů: 30
3. T–typ: `aaaacbbbbd`, `aaaacd`, celkový počet řetězců do délky řetězce 10 symbolů: 25
4. P–typ: `aaaaaaaaaa`, `aaaacbbbbd`, celkový počet řetězců do délky řetězce 10 symbolů: 65

Pro test automatického přijímání řetězců vygenerujeme následujícími příkazy řetězec pro příslušný typ jazyka sítě:

1. L–typ: `./bp -l l -r rule.txt -f test/sit_test_jazyk.xml -g 11 -o`,
   řetězec: aaaaacbbbbb
2. G–typ: `./bp -l g -r rule.txt -f test/sit_test_jazyk.xml -g 10 -o`,
   řetězrc: aaaaaaaaac
3. T–typ: `./bp -l t -f test/sit_test_jazyk.xml -g 10 -o`,
   řetězec: aaaaaacbbd
4. P–typ: `./bp -l p -f test/sit_test_jazyk.xml -g 15 -o`,
   řetězec: aaaaaaaacbbbbbd

Test automatického přijímání řetězců s tisknutím výpočetní posloupnosti:

1. L–typ (viz příloha B.2):
   (a) přijatý: `./bp -l l -r rule.txt -f test/sit_test_jazyk.xml -s aaaaacbbbbb -p`
   (b) nepřijatý: `./bp -l l -r rule.txt -f test/sit_test_jazyk.xml -s aaaaacbbbb -p`
2. G–typ (viz příloha B.3):
   (a) přijatý: `./bp -l g -r rule.txt -f test/sit_test_jazyk.xml -s aaaaaaaaac -p`
   (b) nepřijatý: `./bp -l g -r rule.txt -f test/sit_test_jazyk.xml -s aaaaaaacbd -p`
3. T–typ (viz příloha B.4):
   (a) přijatý: `./bp -l t -f test/sit_test_jazyk.xml -s aaaaaacbbd -p`
   (b) nepřijatý: `./bp -l t -f test/sit_test_jazyk.xml -s aaaaaacbbb -p`
4. P–typ (viz příloha B.5):
   (a) přijatý: `./bp -l p -f test/sit_test_jazyk.xml -s aaaaaaaacbbbbbd -p`
   (b) nepřijatý: `./bp -l p -f test/sit_test_jazyk.xml -s aaaaaadaacbbbbb -p`

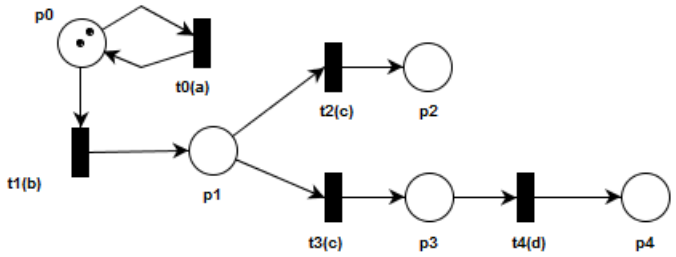
Z příloh lze vidět, že v případě nepřijetí řetězce je vypsána nejdelší posloupnost, která byla provedena, a odpovídající část řetězce.

5.3 Testování ručního zadávání řetězců

V rámci ručního zadávání řetězců je podstatné testovat zejména sítě, které obsahují v nějakém kroku výpočetní posloupnosti možnost provést více přechodů, které jsou ohodnoceny stejným symbolem abecedy sítě. Příkladem jednoduché sítě, která tuto vlastnost splňuje je síť na obrázku 5.3. Tato síť má počáteční značení $M_0 = (2, 0, 0, 0, 0)$ a budeme předpokládat

množinu koncových značení $Q_f = \{(0, 0, 1, 0, 1)\}$.

Na této síti si předvedeme manuální zadávání řetězce, pro třídu jazyka Petriho sítě typu



Obrázek 5.3: Ohodnocená Petriho síť použitá při testování

L, s tím, že necháme předzpracovat řetězec `ba` a ukážeme si použití jednotlivých příkazů jak bylo popsáno v tabulce 4.1 v sekci 4.2.3. Aplikaci spustíme příkazem `./bp -l 1 -r rule.txt -f test/sit_test_konflikt.xml -s ba -i`.

Použití příkazu `clear`:

```
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3,p4)
(2,0,0,0,0)[t1(b)>(1,1,0,0,0)[t0(a)>(1,1,0,0,0)
+-------------------------------------------
clear
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3,p4)
(2,0,0,0,0)
+-------------------------------------------
```

Jak z předcházejícího části výstupu lze vidět, síť se dostane do stavu, který odpovídá nepřečtení žádného symbolu s původním počátečním značením.
Po opětovném spuštění aplikace a zadání symbolu c dojde k vyvolání nabídky:

```
c
Zvolte prechod:
1) t2(c)
2) t3(c)
```

Tento výstup dává na výběr ze dvou proveditelných přechodů. Vlastní výběr je proveden zadáním čísla, kterým je přechod označen.
Po znovuspuštění aplikace se navrátíme do počátečního stavu příkazem `-`, což je zobrazeno další ukázce:

```
+-------------------------------------------
```

```
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3,p4)
(2,0,0,0,0)[t1(b)>(1,1,0,0,0)[t0(a)>(1,1,0,0,0)
+-------------------------------------------
--
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3,p4)
(2,0,0,0,0)
+-------------------------------------------
```

Nyní si zkusíme, zda po opětovném spuštění aplikace a zadání řetězce `bccd` a následném zadání příkazu `exit` dojde k přijetí řetězce sítí.

```
bccd
⋮
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3,p4)
(2,0,0,0,0)[t1(b)>(1,1,0,0,0)[t0(a)>(1,1,0,0,0)
[t1(b)>(0,2,0,0,0)[t2(c)>(0,1,1,0,0)[t3(c)>(0,0,1,1,0)
[t4(d)>(0,0,1,0,1)
+-------------------------------------------
exit
Retezec "babccd"prijat.
```

Pro doplnění si uvedeme použití příkazu exit v počátečním stavu a následné nepřijetí řetězce.

```
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3,p4)
(2,0,0,0,0)[t1(b)>(1,1,0,0,0)[t0(a)>(1,1,0,0,0)
+-------------------------------------------
exit
Retezec "ba"neprijat.
```

Kapitola 6

Závěr

Jádrem této práce bylo navrhnout a implementovat akceptor jazyků Petriho sítě a následně otestovat jeho správnou funkčnost. Akceptor byl navržen pro základní typy jazyků Petriho sítí, tj. typy L, G, T a P s ohodnocením přechodů libovolným symbolem abecedy Petriho sítě ne však prázdným symbolem. Akceptor může příjmout řetězec automaticky nebo se může využít postupného ručního zadávání. Obě uvedené možnosti jsou doprovázeny vizualizací v podobě znázornění výpočetní posloupnosti, kde lze vidět změnu značení při provedení přislušného proveditelného přechodu. Akceptor lze rověž využít jako generátor řetězců jazyka Petriho sítě a tento výstup případně použít jako vstup pro jiné aplikace. Příslušné algoritmy, které zajišťují tyto činnosti byly uvedeny v kapitole 4.2. Funkčnost akceptoru byla otestována na příkladech, u nichž byl znám jejich správný výstup, viz kapitola 5 a příloha B.

Detailní analýza složitosti nebyla provedena (nebylo předmětem zadání), lze však konstatovat, že uvedené algorimty mají polynomiální složitost (vnořené cykly, rekurze), tedy je lze považovat za prakticky použitelné.

Zadávání vstupní sítě je zprostředkováno pomocí XML souboru, který reprezentuje Petriho síť vytvořenou grafickým editorem *PIPE 2*. Vstupní informace jako jsou typ jazyka, Petriho síť a typ funkce, kterou bude akceptor vykonávat, jsou zadávány parametry při spuštění aplikace.

V práci jsou rovněž zmíněny některé známé poznatky týkající se ω–jazyků a Petriho sítí, které jsou schopné specifikovat některé ω–jazyky.

Do budoucna by se aplikace mohla rozšířit o algoritmy syntézy Petriho sítí na základě vlastností jazyků Petriho sítí (sjednocení, průnik, atd.). Další výzkum by se mohl ubírat směrem k Petriho sítím specifikujícím ω–jazyky a rozpoznávání takovýchto sítí.

Literatura

[1] *CMarkup documentation.*
URL <http://www.firstobject.com/dn_markupmethods.htm>

[2] *Platform Independent Petri net Editor 2.*
URL <http://pipe2.sourceforge.net/>

[3] Bokr, J.; Jáneš, V.; Jánešová, M.: Recognizing of Language by an Acceptor or a Petri Net. In *n Proceedings of the Work in Progress held in connection with SEAA the 34Th EUROMICRO Conference on Software Engineering and Advanced Applications and DSD 2008 the 11th EUROMICRO Conference on Digital System Design*, Linz: J. Kepler University, 2008, ISBN 978-3-902457-20-3, s. 1–2.

[4] Češka, M.: *Petriho sítě.* Akademické nakladatelství CERM Brno, říjen 1994, ISBN 80-85867-35-4.

[5] Češka, M.; Vojnar, T.; Marek, V.; aj.: Petriho sítě PES studijní opora. prosinec 2009.
URL <http://www.fit.vutbr.cz/study/courses/PES/public/Pomucky/PES_opora.pdf>

[6] Češka, M.; Vojnar, T.; Smrčka, A.: Teoretická informatika TIN Studijní opora. září 2010.
URL
<http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf>

[7] Gaubert, S.; Giua, A.; Giua, A.: Petri Net Languages and Infinite Subsets of N^m.
URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.3764>

[8] Linz, P.: *An Introduction to Formal Languages and Automata.* USA: Jones and Bartlett Publishers, Inc., třetí vydání, 2001, ISBN 0763714224.

[9] Perrin, D.; Pin, J.-E.: *Infinite words: automata, semigroups, logic and games*, ročník 141. Academic Press, 2004.

[10] Peterson, J. L.: Petri Nets. *ACM Computing Surveys*, ročník 9, 1977: s. 223–252.
URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.137.6622>

[11] Staiger, L.; luther-universitat Halle-wittenberg, M.; Staiger, L.: ω-Languages. 1997.
URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.4015>

[12] Valk, R.: Infinite behaviour of Petri nets. *Theoretical Computer Science*, ročník 25, č. 3, 1983: s. 311 – 341, ISSN 0304-3975, doi:10.1016/0304-3975(83)90115-9.
URL
<http://www.sciencedirect.com/science/article/pii/0304397583901159>

[13] Valk, R.; Vidal-Naquet, G.: Petri nets and regular languages. *Journal of Computer and System Sciences*, ročník 23, č. 3, 1981: s. 299 – 325, ISSN 0022-0000, doi:10.1016/0022-0000(81)90067-2.
URL
<http://www.sciencedirect.com/science/article/pii/0022000081900672>

[14] Yamasaki, H.: Logical Characterization of Petri Net ω-Languages. 1999.
URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.311>

Příloha A

Přehled použitých symbolů a zkratek

$\mathbf{N}^+$	množina celých nezáporných čísel
$\emptyset$	prázdná množina
ε	prázdný řetězec, prázdný symbol
ω	suprémum množiny $\mathbf{N}^+$
a	symbol
$\{a\}$	jednoprvková množina obsahující symbol a
w	řetězec
$\|w\|$	délka řetězce
${}^\bullet x$	vstupní množina prvku x
$x^\bullet$	výstupní množina prvku x
$A \setminus B$	množinový rozdíl
P	množina míst
T	množina přechodů
F	toková relace
W	ohodnocení hran grafu sítě
K	kapacita míst
M	značení sítě
M_0	počáteční značení sítě
$[M\rangle$	množina dosažitelných značení ze značení M
$[t\rangle$	provedení přechodu t
δ	přechodová funkce Petriho sítě
λ	ohodnocení přechodů symboly abecedy
Σ	abeceda
Σ^*	množina konečných řetězců
Σ^ω	množina řetězců neomezené délky
$L_1 \cdot L_2$	konkatenace jazyků
$L_1 \cup L_2$	sjednocení jazyků
$L_1 \cap L_2$	průnik jazyků
$L_1 \parallel L_2$	paralelní kompozice jazyků

L^+	pozitivní iterace jazyka
L^*	iterace jazyka
L^ω	nekonečná iterace jazyka
L^R	reverze jazyka
$L_1 \subseteq L_2$	L_1 je podmnožinou L_2
$\underline{N}$	matice změn
$\underline{F}$	toková matice
XML	obecný značkovací jazyk

Příloha B

Výstupy testů

B.1 Test na správné načtení sítě

Spuštění aplikace s parametry `./bp -f test/test_matrix.xml -t -p`:

```
Abeceda site A = {a, b, c}
+-------------------------------------------
Matice zmen
+-------------------------------------------
     t0(a)  t1(b)  t2(c)
 p0  0      0      0
 p1  -1     2      0
 p2  -1     1      -1
 p3  0      -1     1
+-------------------------------------------

+-------------------------------------------
Tokova matice
+-------------------------------------------
     t0(a)  t1(b)  t2(c)
 p0  (1,1)  (0,0)  (0,0)
 p1  (1,0)  (0,2)  (0,0)
 p2  (1,0)  (0,1)  (1,0)
 p3  (0,0)  (1,0)  (0,1)
+-------------------------------------------
```

B.2 Test na přijetí a nepřijetí řetězce jazyka typu L

Spuštěno příkazem: `./bp -l l -r rule.txt -f test/sit_test_jazyk.xml -s aaaaacbbbbb -p`

```
Abeceda site A = {a, b, c, d}
Retezec "aaaaacbbbbb"prijat
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
```

```
(p0,p1,p2,p3)
(1,0,0,0)[t0(a)>(1,1,0,0)[t0(a)>(1,2,0,0)[t0(a)>(1,3,0,0)
[t0(a)>(1,4,0,0)[t0(a)>(1,5,0,0)[t2(c)>(0,5,1,0)
[t1(b)>(0,4,1,0)[t1(b)>(0,3,1,0)[t1(b)>(0,2,1,0)
[t1(b)>(0,1,1,0)[t1(b)>(0,0,1,0)
+-------------------------------------------
```

Spuštěno příkazem: `./bp -l l -r rule.txt -f test/sit_test_jazyk.xml -s aaaaacbbbb -p`

```
Abeceda site A = {a, b, c, d}
Retezec neprijat, nejdelsi prefix: "aaaaacbbbb"
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3)
(1,0,0,0)[t0(a)>(1,1,0,0)[t0(a)>(1,2,0,0)[t0(a)>(1,3,0,0)
[t0(a)>(1,4,0,0)[t0(a)>(1,5,0,0)[t2(c)>(0,5,1,0)
[t1(b)>(0,4,1,0)[t1(b)>(0,3,1,0)[t1(b)>(0,2,1,0)
[t1(b)>(0,1,1,0)
+-------------------------------------------
```

B.3 Test na přijetí a nepřijetí řetězce jazyka typu G

Spuštěno příkazem: `./bp -l g -r rule.txt -f test/sit_test_jazyk.xml -s aaaaaaaaac -p`

```
Abeceda site A = {a, b, c, d}
Retezec "aaaaaaaaac"prijat
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3)
(1,0,0,0)[t0(a)>(1,1,0,0)[t0(a)>(1,2,0,0)[t0(a)>(1,3,0,0)
[t0(a)>(1,4,0,0)[t0(a)>(1,5,0,0)[t0(a)>(1,6,0,0)
[t0(a)>(1,7,0,0)[t0(a)>(1,8,0,0)[t0(a)>(1,9,0,0)
[t2(c)>(0,9,1,0)
+-------------------------------------------
```

Spuštěno příkazem: `./bp -l g -r rule.txt -f test/sit_test_jazyk.xml -s aaaaaaacbd -p`

```
Abeceda site A = {a, b, c, d}
Retezec neprijat, nejdelsi prefix: "aaaaaaacbd"
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3)
(1,0,0,0)[t0(a)>(1,1,0,0)[t0(a)>(1,2,0,0)[t0(a)>(1,3,0,0)
[t0(a)>(1,4,0,0)[t0(a)>(1,5,0,0)[t0(a)>(1,6,0,0)
[t0(a)>(1,7,0,0)[t2(c)>(0,7,1,0)[t1(b)>(0,6,1,0)
```

```
[t3(d)>(0,6,0,1)
+-----------------------------------------
```

B.4 Test na přijetí a nepřijetí řetězce jazyka typu T

Spuštěno příkazem: `./bp -l t -f test/sit_test_jazyk.xml -s aaaaaacbbd -p`

```
Abeceda site A = {a, b, c, d}
Retezec "aaaaaacbbd"prijat
+-----------------------------------------
Vypocetni posloupnost site (firing sequence)
+-----------------------------------------
(p0,p1,p2,p3)
(1,0,0,0)[t0(a)>(1,1,0,0)[t0(a)>(1,2,0,0)[t0(a)>(1,3,0,0)
[t0(a)>(1,4,0,0)[t0(a)>(1,5,0,0)[t0(a)>(1,6,0,0)
[t2(c)>(0,6,1,0)[t1(b)>(0,5,1,0)[t1(b)>(0,4,1,0)
[t3(d)>(0,4,0,1)
+-----------------------------------------
```

Spuštěno příkazem: `./bp -l t -f test/sit_test_jazyk.xml -s aaaaaacbbb -p`

```
Abeceda site A = {a, b, c, d}
Retezec neprijat, nejdelsi prefix: "aaaaaacbbb"
+-----------------------------------------
Vypocetni posloupnost site (firing sequence)
+-----------------------------------------
(p0,p1,p2,p3)
(1,0,0,0)[t0(a)>(1,1,0,0)[t0(a)>(1,2,0,0)[t0(a)>(1,3,0,0)
[t0(a)>(1,4,0,0)[t0(a)>(1,5,0,0)[t0(a)>(1,6,0,0)
[t2(c)>(0,6,1,0)[t1(b)>(0,5,1,0)[t1(b)>(0,4,1,0)
[t1(b)>(0,3,1,0)
+-----------------------------------------
```

B.5 Test na přijetí a nepřijetí řetězce jazyka typu P

Spuštěno příkazem: `./bp -l p -f test/sit_test_jazyk.xml -s aaaaaaaacbbbbbd -p`

```
Abeceda site A = {a, b, c, d}
Retezec "aaaaaaaacbbbbbd"prijat
+-----------------------------------------
Vypocetni posloupnost site (firing sequence)
+-----------------------------------------
(p0,p1,p2,p3)
(1,0,0,0)[t0(a)>(1,1,0,0)[t0(a)>(1,2,0,0)[t0(a)>(1,3,0,0)
[t0(a)>(1,4,0,0)[t0(a)>(1,5,0,0)[t0(a)>(1,6,0,0)
[t0(a)>(1,7,0,0)[t0(a)>(1,8,0,0)[t2(c)>(0,8,1,0)
[t1(b)>(0,7,1,0)[t1(b)>(0,6,1,0)[t1(b)>(0,5,1,0)
[t1(b)>(0,4,1,0)[t1(b)>(0,3,1,0)[t3(d)>(0,3,0,1)
+-----------------------------------------
```

Spuštěno příkazem: `./bp -l p -f test/sit_test_jazyk.xml -s aaaaaadaacbbbbb -p`

```
Abeceda site A = {a, b, c, d}
Retezec neprijat, nejdelsi prefix: "aaaaaa"
+-------------------------------------------
Vypocetni posloupnost site (firing sequence)
+-------------------------------------------
(p0,p1,p2,p3)
(1,0,0,0)[t0(a)>(1,1,0,0)[t0(a)>(1,2,0,0)[t0(a)>(1,3,0,0)
[t0(a)>(1,4,0,0)[t0(a)>(1,5,0,0)[t0(a)>(1,6,0,0)
+-------------------------------------------
```

Příloha C

Obsah přiloženého CD

Přiložené CD obsahuje zdrojové kódy v jazyce C/C++, text práce ve formátu PDF, zdrojové kódy textu pro systém LaTeXa soubor readme.txt s popisem ovládání aplikace. Dále jsou obsaženy XML soubory se sítěmi použitými a soubory s koncovými značeními, které byly použity pro testování.

Adresář	Popis
src	zdrojové kódy aplikace
latex	zdrojové kódy textu bakalářské práce pro LaTeX
test	použité XML soubory se vstupními Petriho sítěmi a soubory s koncovými značeními
projekt.pdf	text bakalářské práce
readme.txt	soubor s popisem ovládání aplikace
bp	spustitelny binarni soubor (prelozen na serveru merlin)

Tabulka C.1: Adresářová struktura na CD